TBRootAna Maintainers Guide v 0.2

Ian Gable
Department of Physics and Astronomy
University of Victoria
email: igable@uvic.ca

September 26, 2003

Contents

1 Introduction

This manual explains the basic procedures for maintaining the GNU compliant Package TBRootAna. It makes no effort to explain how to use TBRootAna for the purpose[1] it was designed. Information on how to use the package is available from the TBRootAna website[1].

The manual briefly covers all the tools required to maintain the package. These tools include, the source control system, CVS (Concurrent Versioning System), the build tools, autoconf and automake, the documentation system, Doxygen. Procedures are outlined for building the package and making a release.

The procedure for making a release of the Stand Alone Geometry package is also detailed.

2 CVS

The CVS (Concurrent Versioning System) repository for the TBRootAna code is located on the UVic fate machines. The root of the CVS repository is located in cvs-atlas account. Therefore, the path to the root of the CVS repository from any fate machine is `'/homes/cvs-atlas/cvsroot'`. Any user wishing to access the repository must have the environment variable `'$CVSROOT'` set to the aforementioned path with the command,

```
export $CVSROOT=/homes/cvs-atlas/cvsroot
```

2.1 Basic Commands

To administer TBRootAna there are several basic cvs commands which you need to be familiar with.

2.1.1 checkout

Before anything can be done with the source code it must first be checked out of the cvs repository with the cvs command `checkout`. There are several options which can be passed to `checkout` command which I will highlight with an example:

```
cvs checkout -r v0r17 -P TBRootAna
```

`-r` indicates which particular tagged version of the source code you would like to checkout. If this option is left out the head of the repository is checked out.

`-P` Prunes the empty directories out of the source tree. For example the directory `i386_linux` no longer contains any source code but must remain in the repository so that older versions of the code can be checked out. With the `-P` option this directory is not present in your working directory. It is especially important to use this option when checking out code to make a release (see section 2.4).

[1] TBRootAna was designed for the analysis of 2002 combined Hadronic Endcap Calorimeter, Electromagnetic Endcap Calorimeter Test Beam data.

2.1.2 update

To get changes made on the repository into your local working directory you must use the command `update`. Before being able to add your own changes to the cvs repository you must have updated your code from the repository. In the event that you have made changes to a local file and changes have been made to the repository CVS will attempt to merge the the two differing versions of the file. CVS will have varying degrees of success with the merge, the results of which are indicated on the command line. Here is an example session:

```
[gable@fate-3 TBRootAna]$ cvs -n update
cvs update: Updating .
M mainpage.h
C README
cvs update: Updating docs
cvs update: Updating env
cvs update: Updating geo
cvs update: Updating include
cvs update: Updating src
U src/Control.cxx
cvs update: Updating src/readers
cvs update: Updating src/utilities
```

As you can see CVS tells you what exactly it has done in your local directory. Coming directly from the CVS manual[2], here is the meaning of the letters which proceed file names:

`M` *file* The file is modified in your working directory.

'M' can indicate one of two states for a file you're working on: either there were no modifications to the same file in the repository, so that your file remains as you last saw it; or there were modifications in the repository as well as in your copy, but they were merged successfully, without conflict, in your working directory.

`U` *file* The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source, and for files that you haven't changed but are not the most recent versions available in the repository.

`C` *file* A conflict was detected while trying to merge your changes to file with changes from the source repository. *file* (the copy in your working directory) is now the result of attempting to merge the two revisions; an unmodified copy of your file is also in your working directory, with the name '.#file.revision' where revision is the revision that your modified file started from.

There are several other output possibilities but the above are by far the most common.

2.1.3 commit

Commit is used to put changes you have made back into the repository. Before being able to commit code you must have a version that is up to date with the head of the repository.

Example:

```
cvs commit -m ‘‘Fixed a bug’’ Control.cxx
```

`-m` *“comment”* This is a message to add to the commit log for the particular file in question. If you do not include this option CVS will open an editor for you to put your comments into. The default editor is `vi`. To change the editor used set the environment variable `$CVSEDITOR` to the editor of your choice.

`-n` Show results of carrying out this commit without actually making any changes. The `’-n’` option can be used with any CVS command.

If you do not specify a file to commit CVS will recursively check for modified files in the directory and commit them as necessary. Be careful when using this behavior because it can often result in unintentional commits.

2.1.4 diff

The `diff` command allows you to get a standard unix diff between a file in your working directory and a file in the repository. This can be extremely useful for determining where you may have gone wrong since your last commit.

Example:

```
cvs diff Control.cxx
```

2.1.5 log

The `log` command allows you to view in the complete set of comments given when ever a file has been committed.

Example:

```
cvs log Control.cxx
```

2.1.6 rtag

`rtag` is a command with very serious implications for the repository and cannot be undone without significant effort. Therefore, great care should be exercised when using it. `rtag` attaches a symbolic tag to ALL the files at the head of the repository. For further explanation see section 2.3.

Example: `cvs rtag v0r17 TBRootAna`
General Example: `cvs rtag [`*options*`]` *tag module*

tag A symbolic tag you which to attach to all all files at the head of the repository for a particular module.

module The name of the ’module’ (a directory in cvsroot) which you wish to tag.

2.1.7 import

The command `import` allows for the creation of a new module inside the repository. In general it is used only at the beginning of a new project to bring sources into the repository for the first time. The explanation of this procedure is lengthy and best left to the author of the CVS Manual. Please see section 3.1.1 of the CVS Manual [3].

2.2 Possible Problems

CVS is very powerful tool that does the right thing remarkably often, however there is still potential for the inattentive user to cause problems.

A particularly dangerous situation arises when you need perform a critical bug fix in the middle of extensive modifications that are not ready to be committed to the repository.

The proper procedure here is:

1. Checkout the head of the code in a new working area.
2. Make the bug fix.
3. Commit it.
4. Return to the working area where you were making your extensive modifications.
5. Perform an update such that the bug fix is merged into your code.

A faulty, yet tempting, procedure goes as follows:

1. Make a backup copy of the file your working file by renaming it.
2. Perform an update to bring a fresh copy of the file from the repository.
3. Make the bug fix.
4. Commit it.
5. Copy your backup copy over the file you just bug fixed.
6. Perform an update.

This is incorrect because CVS will merge out the changes you have just made to fix the bug because it believes that you are working on the latest version when in reality you are not. When you go to commit your extensive changes you will reintroduce your bug.

2.3 Tags

Tags are tools for symbolically labeling a particular set of versions of the files in a repository. The purpose of a tag is to allow you to identify a particular state of the repository that you would like to easily checkout in the future. More general information on the technicalities of tags in available from the CVS manual [4].

TBRootAna uses several important tag conventions:

- Tags should only be applied when the head of the repository is a condition that is considered usable. Tagged version should always compile.

- A minimum of testing must be done before applying a tag to repository. For example the code should compile and execute without crashing, at the very minimum.
- Any code that is released in the form of a tarball should have a corresponding tag in the repository.
- Only one person is responsible for applying tags.
- Tags appear in the form `vNrM`, where `N` is the major version, and `M` is the minor version (example: `v0r17`). All new tags must have `N` or `M` incremented, however, by how much is arbitrary.
- Most importantly, exercise caution when tagging because errors are painful to fix.

2.4 Making a Release

Making a release is that act of adding a new tag to the repository and creating a corresponding tarball which a contains all the files that correspond to that tag.

Before adding a new tag read section 2.3 to familiarize yourself with the tagging procedure. Once a new tag has been created you only need to checkout the code and make it into a tarball.

The step-by-step procedure for creating a release is:

Step 1: Checkout the a tagged version from the repository making sure to remove unused directories with the `-P` flag.
Example:
```
cvs co -r v0r19 -P TBRootAna
```

Step 2: Rename the `TBRootAna` directory to `tbrootana-0.19`

Step 3: Tar and gzip the directory then move it to the releases directory of the TBRootAna website.
Example:
```
tar -cvf tbrootana-0.19.tar tbrootana-0.19
gzip tbrootana-0.19.tar
mv tbrootana-0.19.tar.gz ~web-atlas/hec-emec/tbrootana/releases/.
```

2.5 Email Notification

When a commit is made to the repository email is automatically sent to all the email address contained in the file `cvs-atlas/notify/notify.txt`. The mail contains information about who made what changes and a link to the CVSweb (see section 2.6). This is accomplished through a Perl SMTP (Simple Mail Transfer Protocol) script located in,

`~cvs-atlas/cvsroot/CVSROOT/logger`.

Each time a commit is made the script,

`~cvs-altas/cvsroot/CVSROOT/loginfo`

is called by CVS. The `loginfo` script then in turn calls `logger`.

The modification of the SMTP script is not recommended for novice Perl programmers for two reasons. Firstly, an error in the script could cause large amounts of unwanted email to be sent to the people in the notify list. Secondly, an error could cause a large volume of traffic on the campus SMTP server smtp.uvic.ca.

2.6 CVSweb

CVSweb is a 'cgi' script which runs on dragon.phys.uvic.ca which access and displays the information contained in the repository in a very useful web browsable form. The Atlas network administrator Jan Van Uytven is responsible for the the maintenance of CVSweb because it requires root access to the principal web server.

3 GNU Autotools

The GNU autotools consist of three programs, autoconf, automake, and libtool. Only autoconf and automake are used by TBRootAna.

The purpose of autoconf and automake is to simplify the building of software and offload the hassles of constructing a portable Makefile from the maintainer of an individual package to the maintainer of autoconf and automake. These tools are extremely powerful and can be extremely complex. Therefore, their description here will be limited to exactly how they are applied to TBRootAna. For extensive documentation see the GNU Manuals[5].

Automake and autoconf require several files. A file called `configure.in` must be placed in the base directory of the package along with a file called `Makefile.am`. Each sub directory of the package must also contain a `Makefile.am`, as automake acts recursively on the package.

`configure.in` is read by both autoconf an automake to determine what actions to take on the package. The syntax of the of the file is rather cryptic and written in language called 'm4'. Luckily few modifications to this file need ever be made. It only needs to be touched when a new directory which contains source code is added to TBRootAna. Say you were to add the directory `TBRootAna/src/foo`. You will need to change the `configure.in` line:

```
AC\_OUTPUT(Makefile src/Makefile src/readers/Makefile \
      src/utilities/Makefile)
```

to:

```
AC\_OUTPUT(Makefile src/Makefile src/readers/Makefile \
  src/utilities/Makefile src/foo/Makefile)
```

The syntax of the `Makefile.am` is much more friendly. These files need to be modified when ever a new source file is added to the repository.

```
SUBDIRS = readers utilities

INCLUDES = -I$(top_srcdir)/include \
        -I$(ROOTSYS)/include

CXX = g++

bin_PROGRAMS = tbrootana


tbrootana_SOURCES =  EmecCell.cxx \
        TBID.cxx \
        HecCell.cxx \
        HecEvent.cxx \
        Event.cxx \
        Headers.cxx \
        Geometry.cxx \
        SystemAlg.cxx \
        UserAlg.cxx \
        App.cxx \
        Control.cxx \
        $(top_srcdir)/include/EmecCell.h \
        $(top_srcdir)/include/TBID.h \
        $(top_srcdir)/include/HecCell.h \
        $(top_srcdir)/include/HecEvent.h \
        $(top_srcdir)/include/Event.h \
        $(top_srcdir)/include/Headers.h \
        $(top_srcdir)/include/Geometry.h \
        $(top_srcdir)/include/SystemAlg.h \
        $(top_srcdir)/include/UserAlg.h \
        $(top_srcdir)/include/App.h \
        $(top_srcdir)/include/Control.h

# ROOTSYS = /net/cern/root-3.05

LDADD = $(top_srcdir)/src/readers/libreaders.a \
        $(top_srcdir)/src/utilities/libutilities.a \
        -L$(ROOTSYS)/lib \
        -lCore \
        -lCint \
        -lHist \
        -lGraf \
        -lGraf3d \
        -lGpad \
        -lTree \
        -lRint \
        -lPostscript \
        -lMatrix \
        -lPhysics \
        -lGui \
        -lGeom

tbrootana_LDFLAGS = -lm -ldl -lpthread -rdynamic
```

To understand how to modify the `Makefile.am` we will examine the file line by line referring to the code listing on the previous page.

`SUBDIRS` A list of all the sub directories where automake must act.

`INCLUDES` A list of all the places the compiler must look to find the include files for the program. `$(ROOTSYS)` is an environment variable that will be read in at compile time. The environment variable `$(top_srcdir)` is created by automatically at compile time.

`CXX` The compiler to use.

`bin_PROGRAMS` The name of the final executable.

`tbrootana_SOURCES` A complete list of all the cxx files contained in the directory and their associated header files.

`LDADD` A list of the libraries the compiler should use for linking.

`tbrootana_LDFLAGS` Flags to pass to the linker.

3.1 Running autoconf and automake

After the appropriate files have been created actually running automake and autoconf is quite simple:

```
[gable@fate-2 gable]$cd TBRootna
[gable@fate-2 TBRootAna]$automake -a
[gable@fate-2 TBRootAna]$autoconf
```

The '`-a`' flag causes automake to automatically calculates the dependencies among the files. The output of the commands is a `Makefile.in` in each of the directories of TBRootAna and a script called `configure` in the base directory.

Any user can now execute the script `configure` which processes the `Makefile.in` files to produce `Makefile` files which are specific to the particular platform. Now the user types '`make`' to build TBRootAna. In summary:

```
[gable@fate-2 TBRootAna]$./configure
[gable@fate-2 TBRootAna]$make
```

3.2 What should be committed to the Repository

Only files which are not generated by the `configure` script should be committed. Files which are created by automake and autoconf should be committed because the end user should not have to use automake and autoconf. In other words do not commit `Makefiles`.

4 Doxygen

Doxygen is an automatic code documentation system used by TBRootAna. Doxygen strips out the inline comments of a particular style from the source code and creates a coherent set of documentation for the package. The documentation created comes in several forms but the form used by TBRootAna is html.

Doxygen works by reading a configuration file that contains information about were the source code is and where you want the documentation to go. Since the documentation is web based the configuration file, called `doxygen.config`, is located in `~/web-atlas/scripts/regenerateDoxygen/`. To configure doxygen edit the configuration as appropriate (file internally documented). Then execute the command:

```
doxygen doxygen.config
```

To do the whole process automatically execute the command:

```
redox
```

from anywhere. This aliased command will execute a Perl script located in the same directory as the configuration file.

5 File Listing

5.1 Top Level Directory

`aclocal.m4` A standard file required by autoconf. If you were to write custom configuration macros they would go in this file.

`AUTHORS` File required in a GNU Package. Contains a list of the authors of the code.

`ChangeLog` A file containing a log of all the changes made to TBRootAna. This file has become rather deprecated as of August 2003.

`configure.in` File required by automake and autoconf. It contains a list of the configure macros to be run.

`COPYING` File required in a GNU Package. It contains the user license, which in case of TBRootAna is the GPL (Gnu Public License).

`INSTALL` File required in a GNU Package. It should contain information about how to install the package.

`install-sh` A script to install the package. This script is blank as of August 2003.

`mainpage.h` A file used by doxygen to create the the first page of the doxygen documentation.

`missing` File required in a GNU Package.

`mkinstalldirs` File required in a GNU Packages. Should be a script to create the directories where TBRootAna would be installed. It is blank as of August 2002.

`NEWS` File required in a GNU Package. Should contain current news about the package.

`README` File required in a GNU Package. A standard README file.

5.2 docs directory

`release.notes` Notes on each CVS tag in the repository. Each entry should contain some information about improvements made since the last release.

`TBRootAnaUserGuide.html` The users guide for TBRootAna.

5.3 env directory

This directory should contain bash scripts for setting up the environment before a build takes place.

`cernLxplus` The script for setting up the environment on the lxplus.cern.ch cluster at CERN.

`uvicFates` The script for setting up the environment on the fate machines at UVic.

6 Stand Alone Geometry Package

The Stand Alone Geometry Package takes the Geometry service elements of TBRootAna and bundles them into a package that builds a shared library.

Say you wanted to assemble a release from TBRootAna tag version `v0r17` and StandAloneGeometry tag version `v0r1` do the following:

```
[gable@fate-1 ~]$mkdir temp
[gable@fate-1 ~]cd temp
[gable@fate-1 temp]$cvs co -r v0r1 StandAloneGeometry
[gable@fate-1 temp]$cd StandAloneGeometry
[gable@fate-1 StandAloneGeometry]$./getpackageready v0r17 0.1
```

This will create a tarball called `standalonegeometry-0.1.tar.gz` in the temp directory. This tarball contains all the files necessary for a release. You can move it to the release section of the TBRootAna website.

The script `getpackageready` checks out TBRootAna and then from it assembles the appropriate files for the release. It then creates a root dictionary for the files and makes the tarball.

References

[1] The TBRootAna website:
http://particle.phys.uvic.ca/~web-atlas/atlas/hec-emec/tbrootana/

[2] Cederqvist, Per, "The CVS Manual", version 1.11.6, 2003, p. 121, on web at: http://www.cvshome.org/docs/manual/

[3] Cederqvist, Per, "The CVS Manual", p. 29

[4] Cederqvist, Per, "The CVS Manual", p. 34

[5] The GNU Manuals, on web at: http://www.gnu.org/manual/manual.html